CSE 590

Computer Architecture

Homework/Assignment No:- 1

DESIGN OF 8-BIT NON-PIPELINED PROCESSOR

Author 1 :
Karthikeyan RS
50289080

Author 2:
David Jegan
50290785

Instructor:
Dr.Sridhar

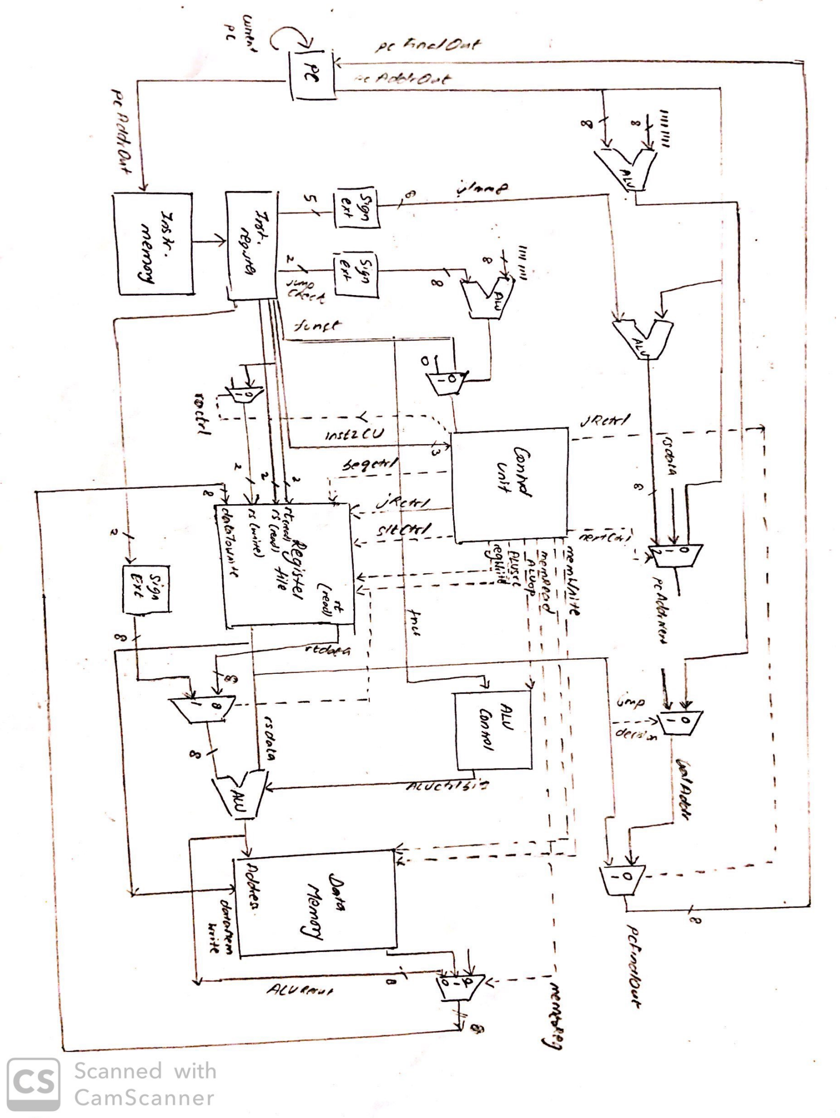
April 16, 2019

Acknowledgement

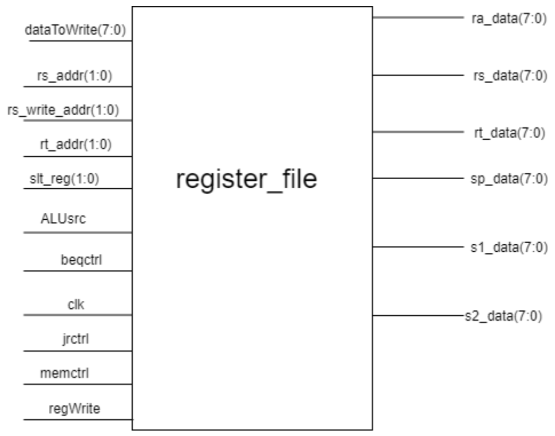
This project is heavily inspired from the Aardvark tutorial and 16 bit single cycle mips processor tutorial in FPGA4Student site. We were able to understand their functionality and tried to replicate the standard outputs.
The basys was inspired from https://www.youtube.com/watch?v=6_GxkslqbcU link of Digilentic.

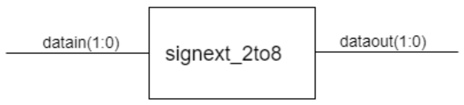
1. https://github.com/brendabrandy/aardvark-8-bit-computer
2. https://www.fpga4student.com/2017/01/verilog-code-for-single-cycle-MIPS-processor.html
3. https://www.youtube.com/watch?v=6_GxkslqbcU
4. https://github.com/Digilent/digilent-xdc/blob/master/Basys-3-Master.xdc

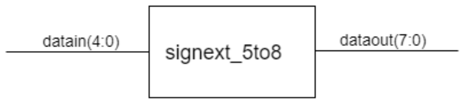
1 Block diagram - Overall

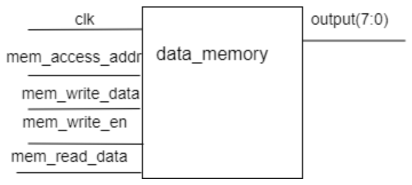


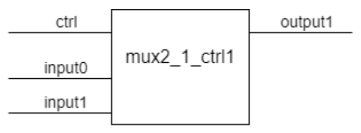
2 Block diagrams - individual components

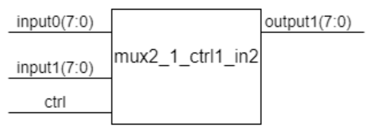


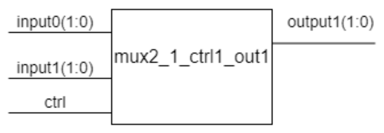


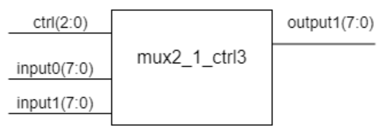


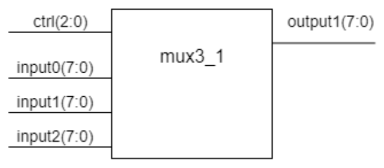


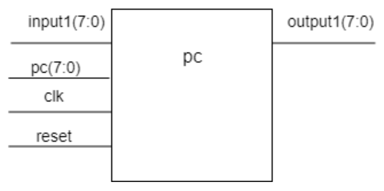

ctrl
input0
input1
mux2_1_ctrl1
output1

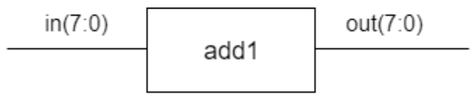

input0(7:0)
input1(7:0)
ctrl
mux2_1_ctrl1_in2
output1(7:0)

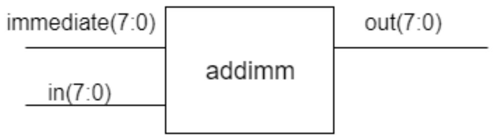

input0(1:0)
input1(1:0)
ctrl
mux2_1_ctrl1_out1
output1(1:0)

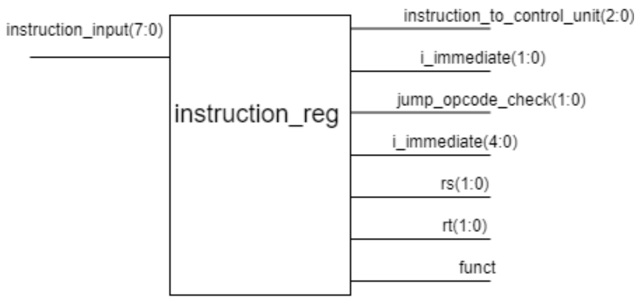

ctrl(2:0)
input0(7:0)
input1(7:0)
mux2_1_ctrl3
output1(7:0)

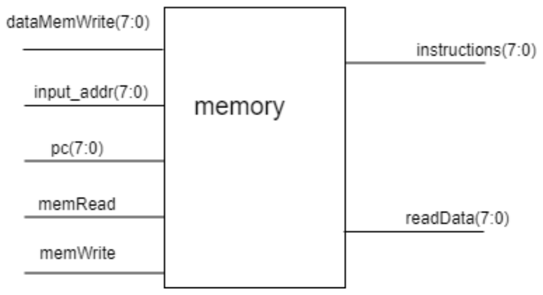

ctrl(2:0)
input0(7:0)
input1(7:0)
input2(7:0)
mux3_1
output1(7:0)

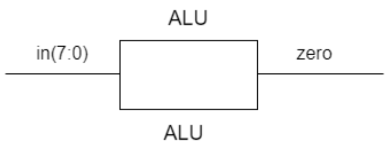

input1(7:0)
pc(7:0)
clk
reset
pc
output1(7:0)

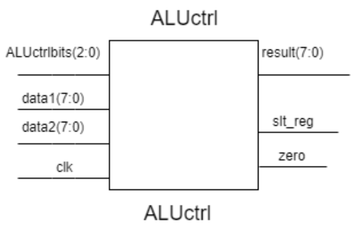

in(7:0)
add1
out(7:0)

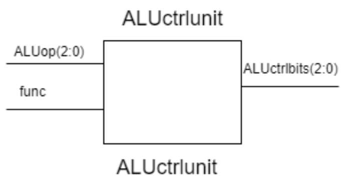

immediate(7:0)
in(7:0)
addimm
out(7:0)

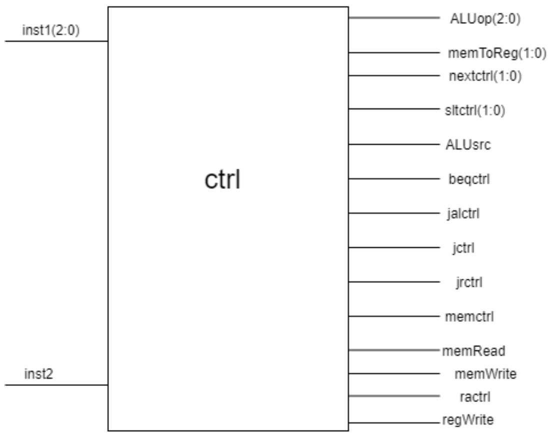

instruction_input(7:0)
instruction_reg
instruction_to_control_unit(2:0)
i_immediate(1:0)
jump_opcode_check(1:0)
i_immediate(4:0)
rs(1:0)
rt(1:0)
funct


dataMemWrite(7:0)
input_addr(7:0)
pc(7:0)
memRead
memWrite
memory
instructions(7:0)
readData(7:0)

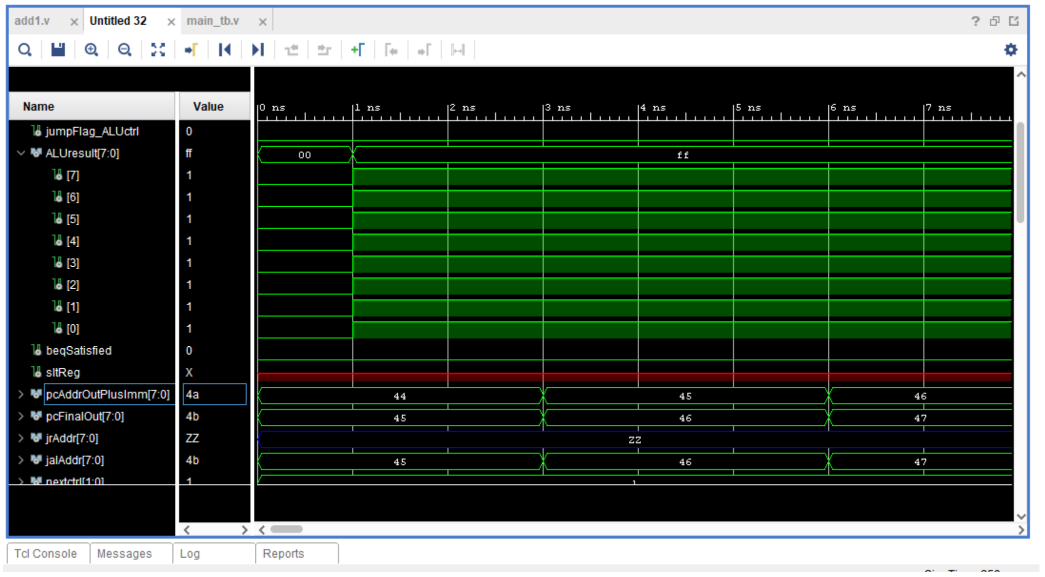

ALU
in(7:0)
zero
ALU


ALUctrl
ALUctrlbits(2:0)
data1(7:0)
data2(7:0)
clk
result(7:0)
slt_reg
zero
ALUctrl

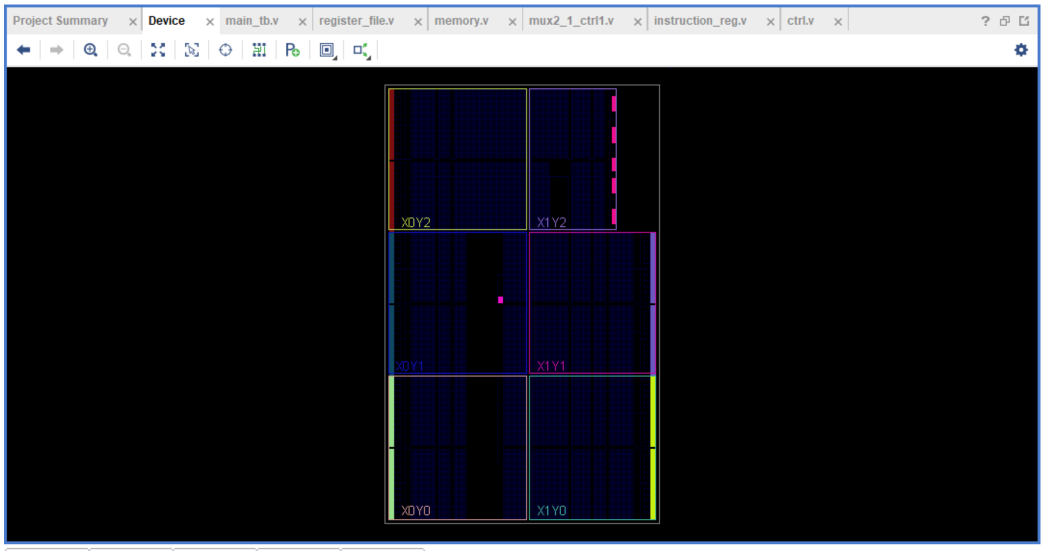

ALUctrlunit
ALUop(2:0)
func
ALUctrlbits(2:0)
ALUctrlunit


inst1(2:0)
inst2
ctrl
ALUop(2:0)
memToReg(1:0)
nextctrl(1:0)
sltctrl(1:0)
ALUsrc
beqctrl
jalctrl
jctrl
jrctrl
memctrl
memRead
memWrite
ractrl
regWrite

3 Simulation results



Name	Value
jumpFlag_ALUctrl	0
ALUresult[7:0]	ff
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	1
[1]	1
[0]	1
beqSatisfied	0
sltReg	X
pcAddrOutPlusImm[7:0]	4a
pcFinalOut[7:0]	4b
jrAddr[7:0]	ZZ
jalAddr[7:0]	4b
nextctrl[1:0]	1



4 Working of each component

Inst Register

1. Input: Instruction input
2. Output:
   (a) J immediate: jump for i type instructin - contains address to jump to
   (b) Instruction to control unit - contains function for trigger in CU
   (c) rt, rs - For R type instructions, regards to the registers
   (d) i immediate- for address in immediate slot
3. Functionality: Decode instruction provided by Instruction memory. It finds the R, J or I inst type and gets opcode from it.

Inst Memory

1. Input: From PC. The size is 256*8 bits
2. Output: To Ins register
3. Functionality: Stores 256 instructions as 1 byte sequence. Input of this will be PCs output and would go for instruction decoder.

Program Counter

1. Input: clock, recursive pc + 1, reset
2. Output: PC sequence to Inst memory
3. Functionality: Fetch every PC sequence

Control Unit

1. Input: opcode, function
2. Output:
   (a) MemtoReg: Flag if Register input data provided from ALU

   (b) RegWrite: Flag if the instruction will update the value of the register.
   (c) MemWrite: Flag if present operation is going to write into memory.
   (d) MemRead: Flag if present operation is going to read from memory.
   (e) jctrl: Flag if jump is needed
   (f) jrctrl: Flag if jump return is needed
   (g) ALUSrc: Flag if an immediate or register data is to be provided to the ALU.
   (h) ALUop: Flag will perform according to the instruction the ALU.
   (i) beqctrl: flag to be selected for a multiplexer to feed the pc.
   (j) nextctrl: flag to be selected for next instruction to feed the pc.

3. Functionality: Generating the control signals according to the opcode and function

Register file

1. Output:
   (a) rt, rs data: return data
   (b) s1, s2 data: internal registers
   (c) sp, ra: stack pointer and return address
2. Input: regWrite, beqctrl, jrctrl, memctrl, ALUsrc, rt addr, rs addr, dataToWrite, slt reg, rs write addr, clk
3. Functionality: The output of the register file will be the register's data according to the input operand.

ALU

1. Input: incoming operation sequence
2. Output: Output operation sequence
3. Functionality: The inputs will be register data and immediate value depending upon the instruction type

ALU ctrl

1. Input: ALU ctrl bits - control bit from the ALU control unit
2. Output: result operation sequence
3. Functionality: The inputs will be register data and immediate value depending upon the instruction type

Memory

1. Input: PC, memread, memwrite, dataMemWrite and input address
2. Output: readData sequence, ROM data
3. Functionality: We read instructions from ROM data and load it to the memory. THen we fetch each instruction and pass it to the system

Sign extension 2 to 8

1. Input: Instr register 2 bits
2. Output: ALU/Mux sequence 8 bits
3. Functionality: The input is extended based on sign and sent to output

Sign extension 5 to 8

1. Input: Instr register 5 bits
2. Output: ALU sequence 8 bits
3. Functionality: The input is extended based on sign and sent to output

MUX 3 to 1

1. Input: Three 8 bit values and a control bit
2. Output: Input 1/2/3 based on ctrl flag
3. Functionality: The output is based on the input flag

MUX 2 to 1 - control 3 bits

1. Input: Two 8 bit values and three bit control

2. Output: Input 1/2 based on ctrl flag
3. Functionality: The output is based on the input flag

MUX 2in to 1 - control flag

1. Input: Two 2 bit values and one flag control
2. Output: Input 1/2 based on ctrl flag
3. Functionality: The output is based on the input flag

MUX 2 to 1 - control flag

1. Input: Two 8 bit values and one flag control
2. Output: Input 1/2 based on ctrl flag
3. Functionality: The output is based on the input flag

Basys 3

1. Input: Clock, btnC, btnU, btnDm btnR, btnL, sw[7:0]
2. Output: led[7:0],Seven Segment Display seg[6:0], an[3:0], dp
3. Functionality: Slide switch inputs (sw) is responsible for inputting from the Basys board. Button are 3 in value and are used for inputing the add and sub commands. dp is the display component

5 Steps

We first ran a C program named Assembler.c to convert the opcodes to binary. Then we stored these binary files inside the memory as ROM. These values is getting referenced by the PC and Instruction register to decode the byte address at each level.

6 Work distribution

Understanding and running Vivado - David and Karthi
Simulation and Basys3 - Karthi and David
Writing .v scripts - David and Karthi

7 References

1. https://github.com/brendabrandy/aardvark-8-bit-computer
2. https://www.fpga4student.com/2017/01/verilog-code-for-single-cycle-MIPS-processor.html
3. https://www.youtube.com/watch?v=6_GxkslqbcU
4. https://github.com/Digilent/digilent-xdc/blob/master/Basys-3-Master.xdc